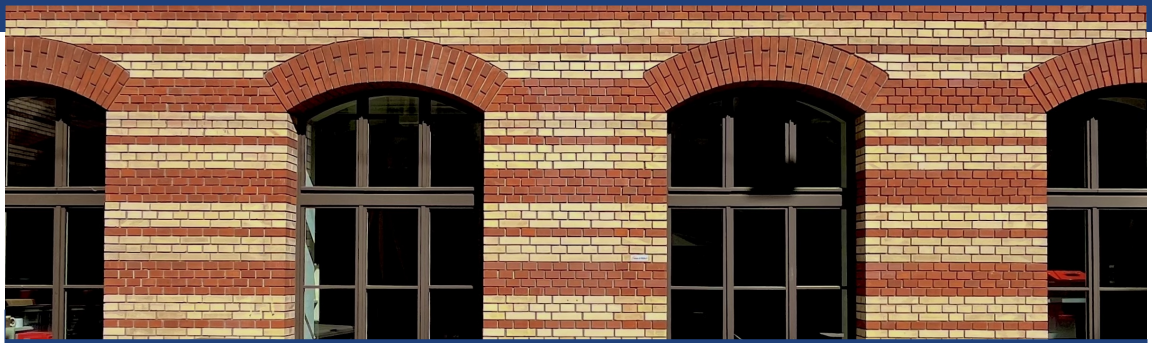


Exercise Session — Computer Science — 07

ASCII Characters, `char`, Recursion

Overview

Today's Plan

(ASCII) Characters in C++ (`char`)
Recursion

1. Feedback regarding **code** expert

General things regarding **code** expert

Any questions regarding **code** expert on your part?

2. (ASCII) Characters in C++ (`char`)

Exercise "Converting Input to UPPER CASE"

Task

1. Consider how best to approach the "Converting Input to UPPER CASE" task on **code** expert

Exercise "Converting Input to UPPER CASE"

Task

1. Consider how best to approach the "Converting Input to UPPER CASE" task on **code** expert
2. Implement (optionally in groups) a solution

Exercise "Converting Input to UPPER CASE"

Task

Write a program that reads a sequence of characters, delimited by the new-line character, as a vector of `char`. Then the program should output the sequence with all lower-case letters changed to UPPER-CASE letters.
To read the sequence you can:

- read a single character from standard input
- insert it into a vector of chars
- repeat until you find a newline character (`\n`)

Please put the code that converts the entire sequence to upper-case and a single character to upper-case into separate functions (you should have at least three functions).
Hint: variables of type char can be treated as numbers

ASCII Table (but you don't really need it!)

"Converting Input to UPPER CASE" — Solution

```
#include <iostream>
#include <vector>
#include <ios>          // not really needed, don't worry about it
```

"Converting Input to UPPER CASE" — Solution

```
// POST: Converts the letter to upper case.
void char_to_upper(char& letter){
    int shift_distance = 'a' - 'A';      // 'a' > 'A' (if conv. to ints)
                                         // distance between the upper
                                         // and lower case numbers
    if('a' <= letter && letter <= 'z'){
        letter -= shift_distance;
    }
}

// POST: Converts all letters to upper-case.
void to_upper(std::vector<char>& letters){
    for(unsigned int i = 0; i < letters.size(); ++i){
        char_to_upper(letters.at(i));
    }
}
```

"Converting Input to UPPER CASE" — Solution

```
std::vector<char> letters;
char ch;

// Step 1: Read input.
do {
    std::cin >> ch;
    letters.push_back(ch);
} while(ch != '\n');

// Step 2: Convert to upper-case.
to_upper(letters);

// Step 3: Output.
for(unsigned int i = 0; i < letters.size(); ++i){
    std::cout << letters.at(i);
}
```

Questions?

3. Recursion

Previous Exam Question

Key data

- Exam: 01.2022 Computer Science (MATH/PHYS/RW)
- Simple recursion task
- Total Points in Exam: 85 points
- Total Time for Exam: 120 minutes
- Points for Task in Exam: 5 points

Previous Exam Question

Key data

- Exam: 01.2022 Computer Science (MATH/PHYS/RW)
- Simple recursion task
- Total Points in Exam: 85 points
- Total Time for Exam: 120 minutes
- Points for Task in Exam: 5 points
- Estimated Time for Task: 7 minutes $= 120 * (5/85)$

Tiny Exam Simulation

- Get into "exam mode" and prepare everything you might need

Tiny Exam Simulation

- Get into "exam mode" and prepare everything you might need
- Open "[Exam 2022.01 (MATH/PHYS/RW)] Compute Series" on **code** expert

Tiny Exam Simulation

- Get into "exam mode" and prepare everything you might need
- Open "[Exam 2022.01 (MATH/PHYS/RW)] Compute Series" on **code** expert
- Implement a (recursive) solution
- Time: 7 Minutes

Previous Exam Question

We want to write a function with the following PRE and POSTs

```
// PRE:   a positive integer n
//
// POST:  returns the n-th number of a series x_n, defined as
//        x_n = 2,                      for n = 1
//        x_n = 1,                      for n = 2
//        x_n = x_(n-1) + x_(n-2),      for n > 2
//
// Example:
//  * n == 1 ~~> 2
//  * n == 2 ~~> 1
//  * n == 3 ~~> 3
```

Previous Exam Question — Solution

Previous Exam Question — Solution

```
// PRE:   a positive integer n
//
// POST:  returns the n-th number of a serie x_n, defined as
//        x_n = 2,                    for n = 1
//        x_n = 1,                    for n = 2
//        x_n = x_(n-1) + x_(n-2),    for n > 2

unsigned int compute_element(unsigned int n) {
    if (n == 1) {
        return 2;
    } else if (n == 2) {
        return 1;
    } else {
        return compute_element(n-1) + compute_element(n-2);
    }
}
```

Questions?

Exercise "Partial Sum"

Task

Write a function that

1. Computes the sum of all natural numbers below (and equal to) n using recursion and returns this value
2. Outputs all the added terms in ascending order (from 0 to n to the console in the same recursive function)

Exercise "Partial Sum"

- Open "Partial Sum" on **code** expert

Exercise "Partial Sum"

- Open "Partial Sum" on **code** expert
- Think about how you would approach the problem with pen and paper

Exercise "Partial Sum"

- Open "Partial Sum" on **code** expert
- Think about how you would approach the problem with pen and paper
- Implement a (recursive) solution (optionally in groups)

"Partial Sum" — Solution

"Partial Sum" — Solution

```
unsigned int partial_sum(const unsigned int n) {
  if (n == 0) {
    return 0;
  } else {
    // print descending
    // std::cout << n << std::endl;

    unsigned int partial = partial_sum(n - 1);

    // print ascending
    std::cout << n << std::endl;

    return n + partial;
  }
}
```

"Partial Sum" — Solution

```
int main() {
  std::cout << "n = ";

  unsigned int n;
  std::cin >> n;

  std::cout << partial_sum(n) << std::endl;

  return 0;
}
```

Questions?

Exercise "Power Function"

Question

How many recursive calls does the following function need to compute x^7?

```
unsigned int power(const unsigned int x, const unsigned int n) {

    if (n == 0){
        return 1;
    } else if (n == 1) {
        return x;
    }

    return x * power(x, n - 1);
}
```

Answer:

Exercise "Power Function"

Question

How many recursive calls does the following function need to compute x^7?

```
unsigned int power(const unsigned int x, const unsigned int n) {

    if (n == 0){
        return 1;
    } else if (n == 1) {
        return x;
    }

    return x * power(x, n - 1);
}
```

Answer: 7

Exercise "Power Function"

- Open "Power Function" on **code** expert

Exercise "Power Function"

- Open "Power Function" on **code** expert
- Think about how you would approach the problem with pen and paper

Exercise "Power Function"

- Open "Power Function" on **code** expert
- Think about how you would approach the problem with pen and paper
- Implement a (recursive) solution (optionally in groups)
- *Hint: This task is a generalization of the task "Multiply with 29" from the first week*

"Power Function" — Solution

```
// POST: result == x^n
unsigned int power (const unsigned int x, const unsigned int n) {
  if(n == 0) {
    return 1;
  } else if(n == 1) {
    return x;
  } else if(n % 2 == 0) {        // case n = 2m for some m in N
    int temp = power(x, n/2);    // temp, to not call the function twice!
    return temp * temp;          // since x^n = x^(2m) = x^m * x^m
  } else {
    return x * power(x, n-1);
  }
}
```

Questions?

The Towers of Hanoi

Struggling with this exercise is a bit of a rite of passage for newbie programmers. It's notoriously difficult if one is not familiar with recursion.

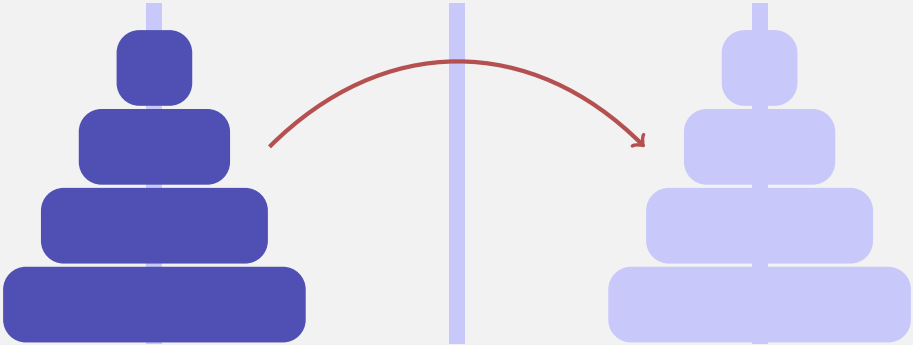
The Towers of Hanoi

Struggling with this exercise is a bit of a rite of passage for newbie programmers. It's notoriously difficult if one is not familiar with recursion.

Experiment: The Towers of Hanoi

left middle right

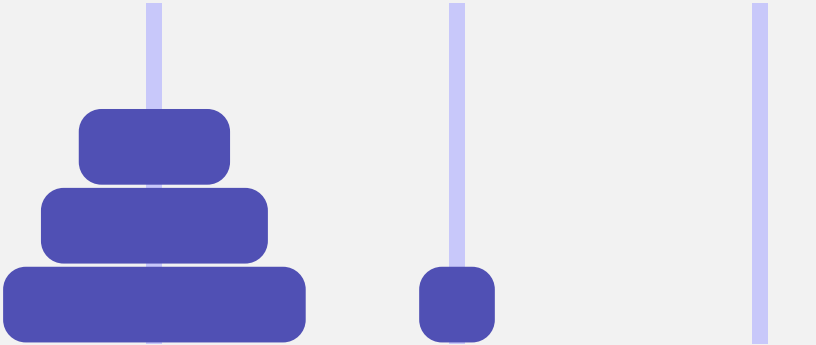
Experiment: The Towers of Hanoi



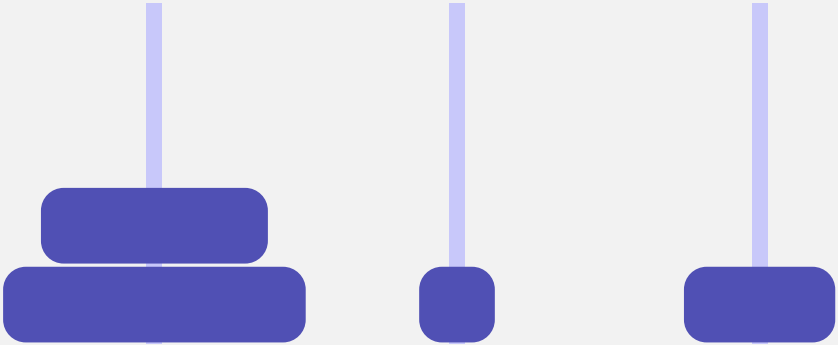
Die Türme von Hanoi - So gehts!

left middle right

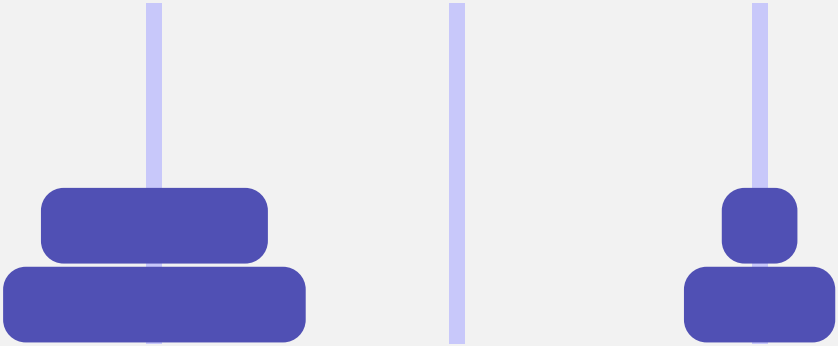
Die Türme von Hanoi - So gehts!



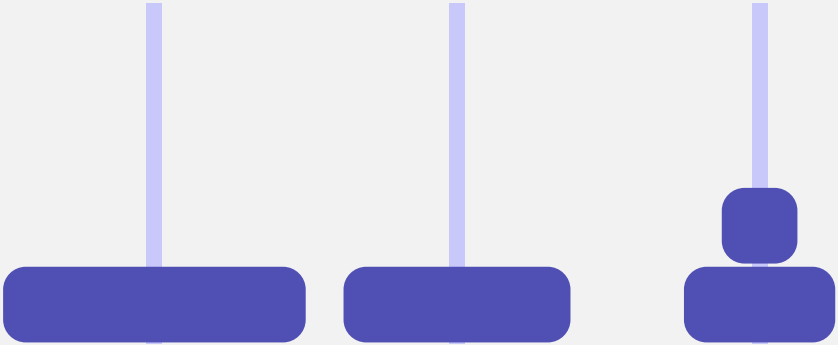
Die Türme von Hanoi - So gehts!



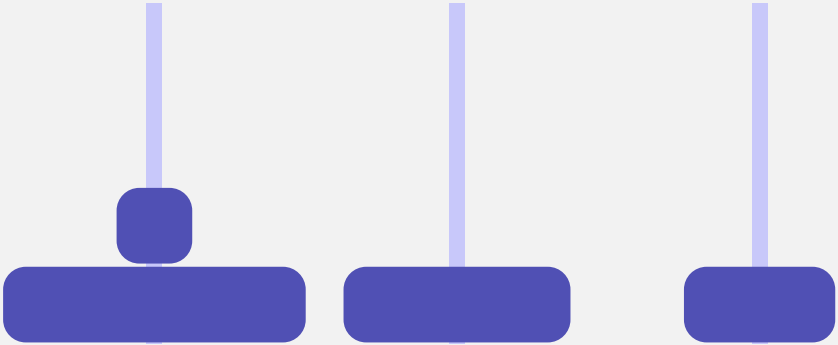
Die Türme von Hanoi - So gehts!



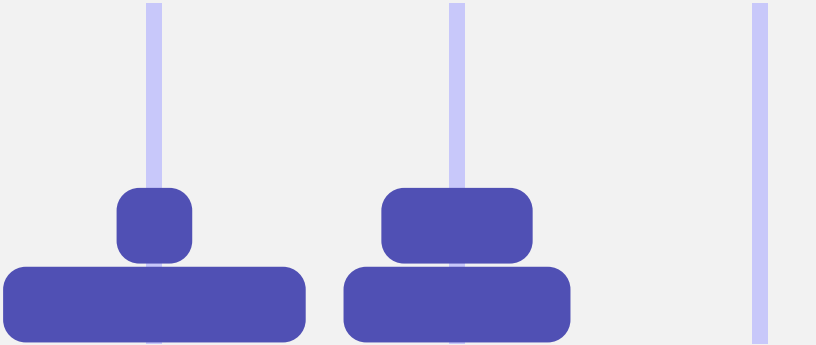
Die Türme von Hanoi - So gehts!



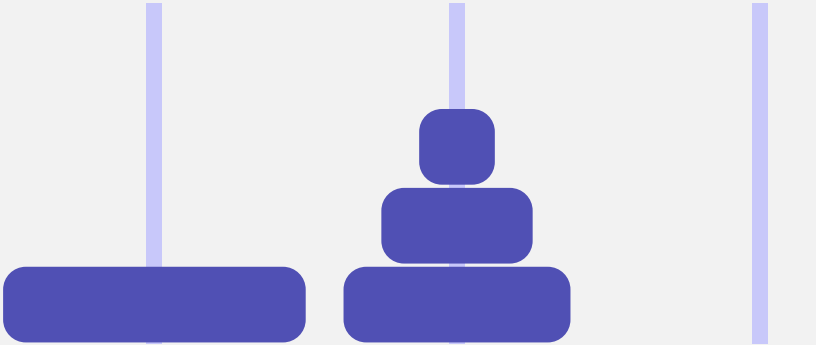
Die Türme von Hanoi - So gehts!



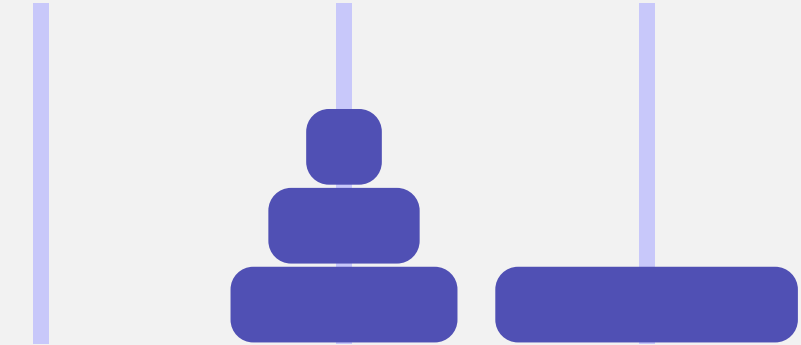
Die Türme von Hanoi - So gehts!



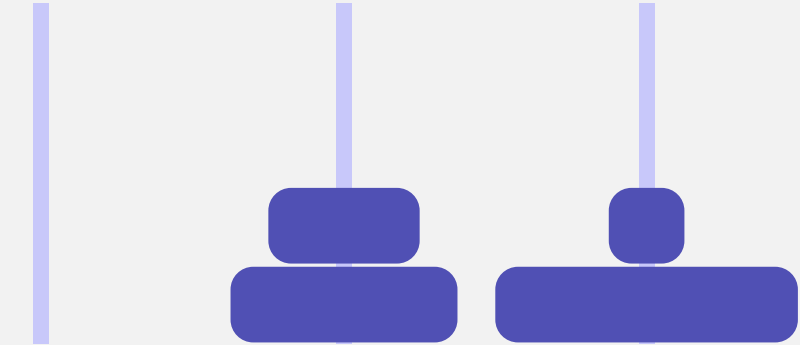
Die Türme von Hanoi - So gehts!



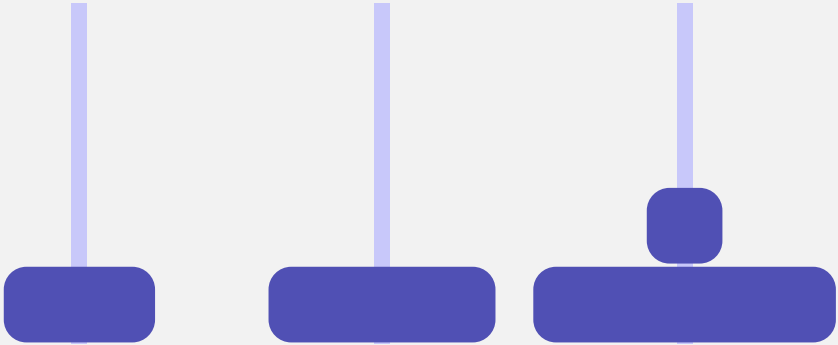
Die Türme von Hanoi - So gehts!



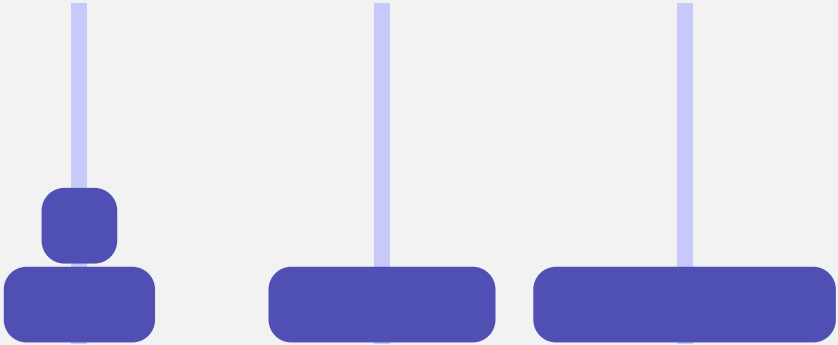
Die Türme von Hanoi - So gehts!



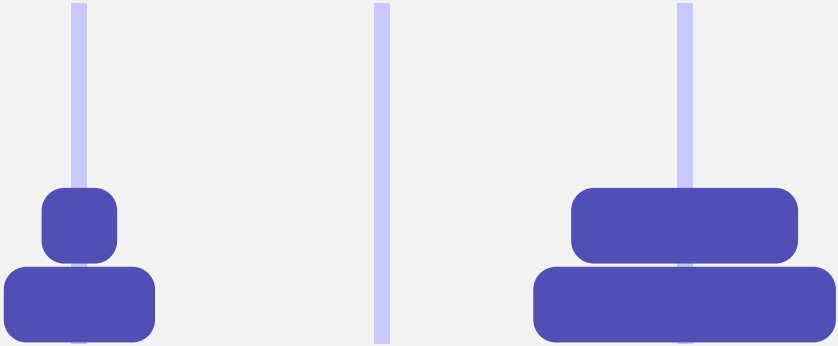
Die Türme von Hanoi - So gehts!



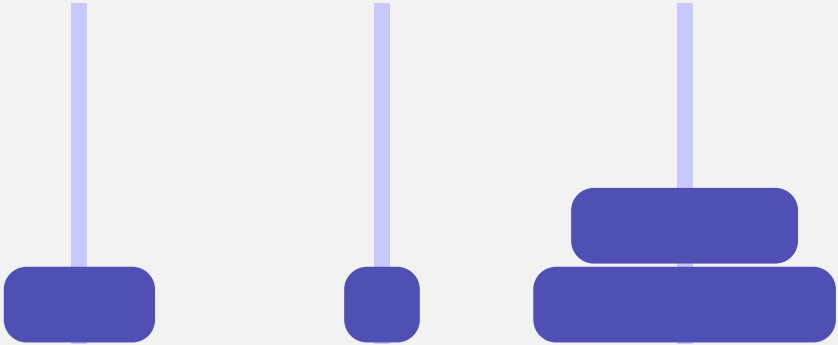
Die Türme von Hanoi - So gehts!



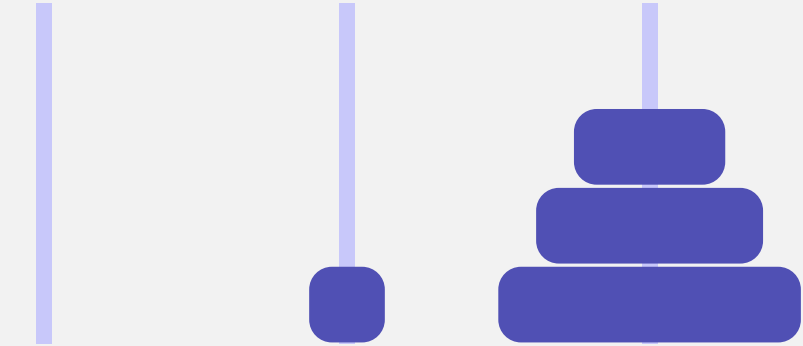
Die Türme von Hanoi - So gehts!



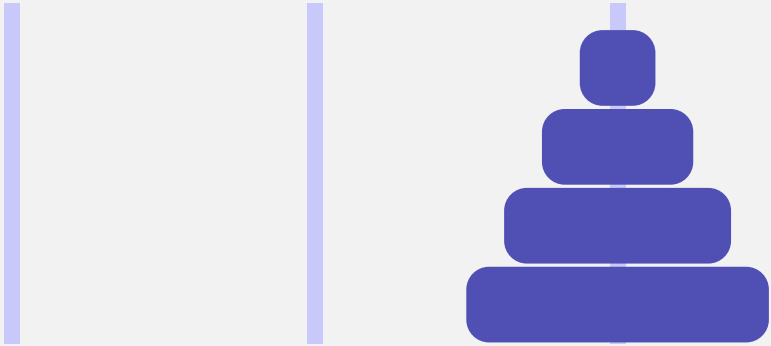
Die Türme von Hanoi - So gehts!



Die Türme von Hanoi - So gehts!



Die Türme von Hanoi - So gehts!

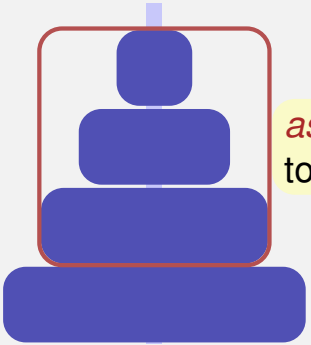


left middle right

The Towers of Hanoi – Recursive Approach

left middle right

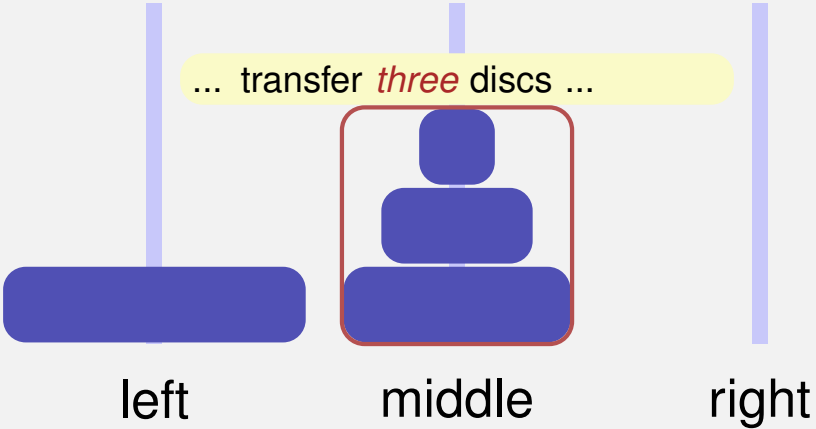
The Towers of Hanoi – Recursive Approach



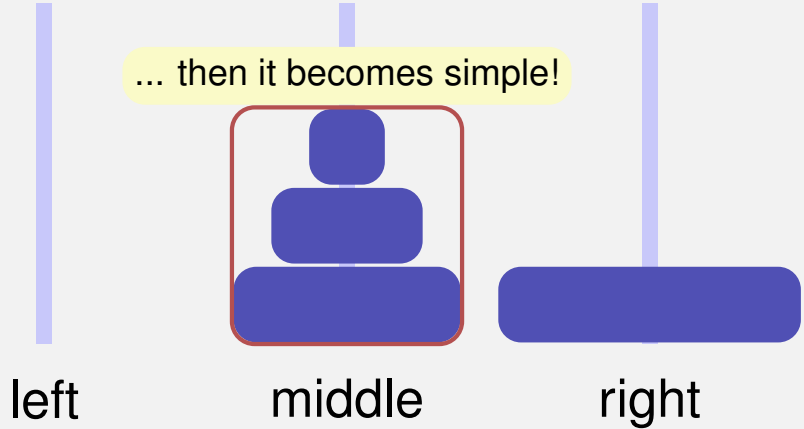
assume we knew how to ...

left middle right

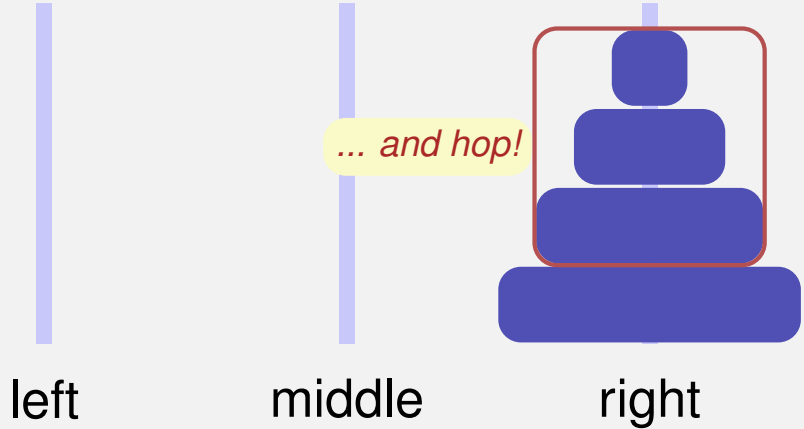
The Towers of Hanoi – Recursive Approach



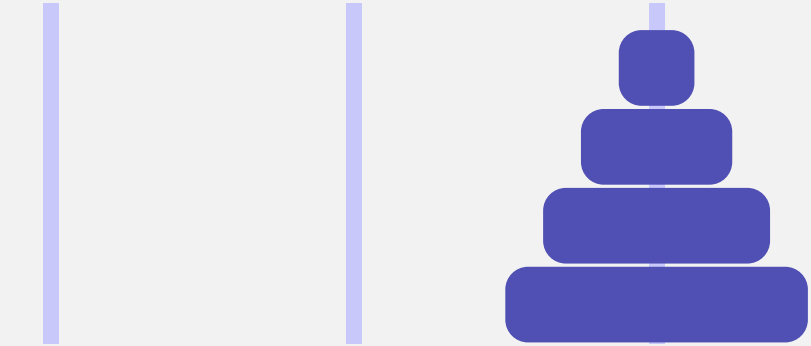
The Towers of Hanoi – Recursive Approach



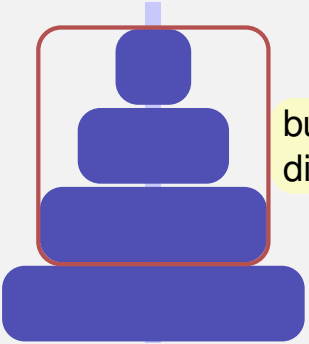
The Towers of Hanoi – Recursive Approach



The Towers of Hanoi – Recursive Approach



The Towers of Hanoi – Recursive Approach



but *how* can we move three discs?

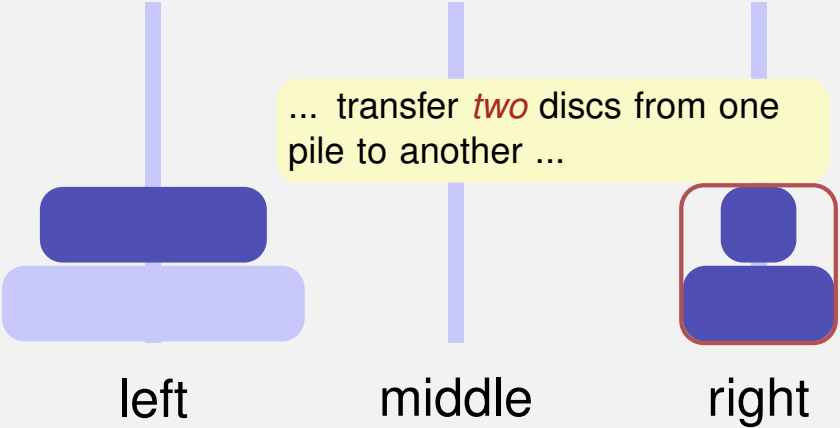
left middle right

The Towers of Hanoi – Recursive Approach

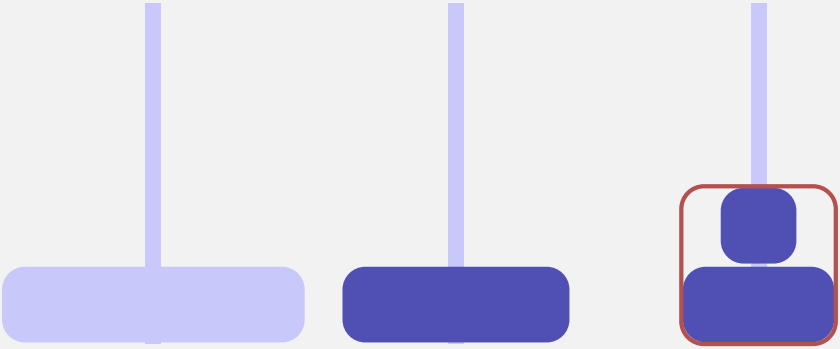
assume we knew how to ...

left middle right

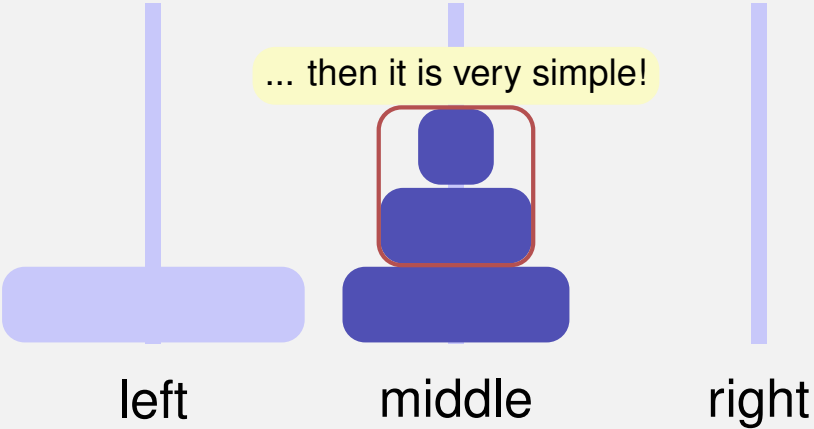
The Towers of Hanoi – Recursive Approach



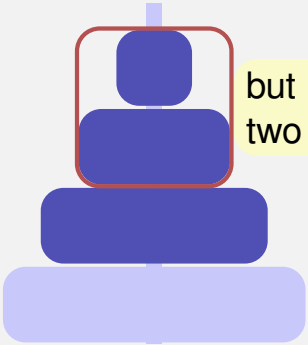
The Towers of Hanoi – Recursive Approach



The Towers of Hanoi – Recursive Approach



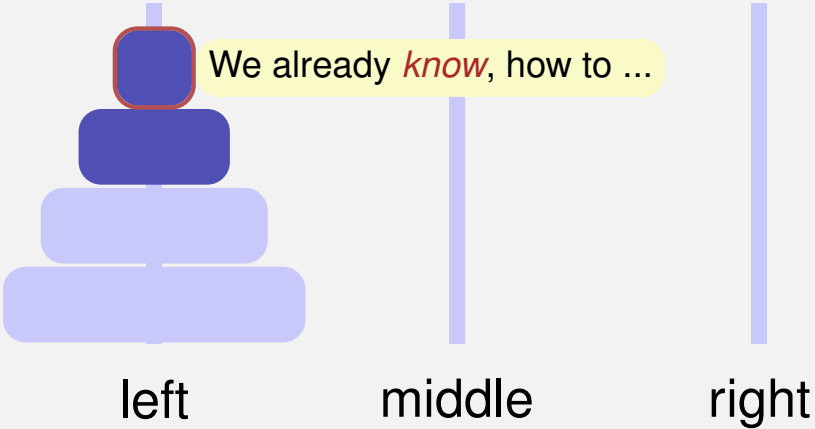
The Towers of Hanoi – Recursive Approach



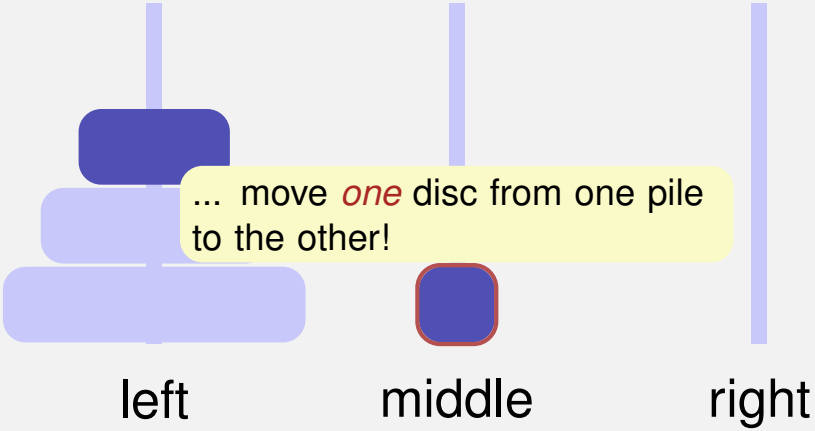
but *how* can we transfer two discs?

left middle right

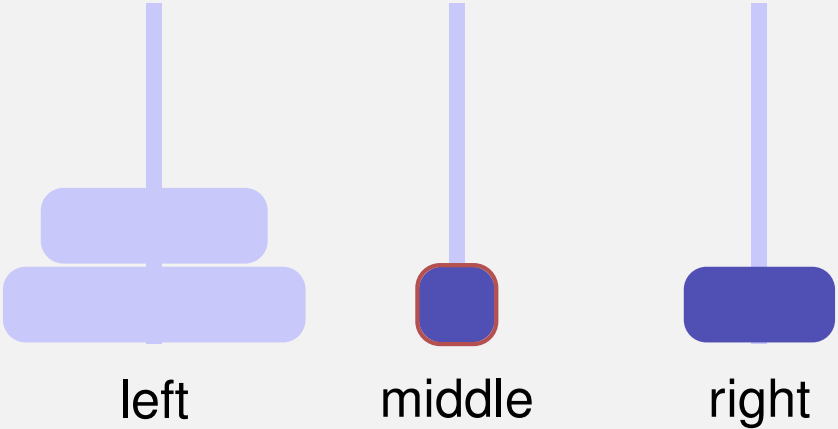
The Towers of Hanoi – Recursive Approach



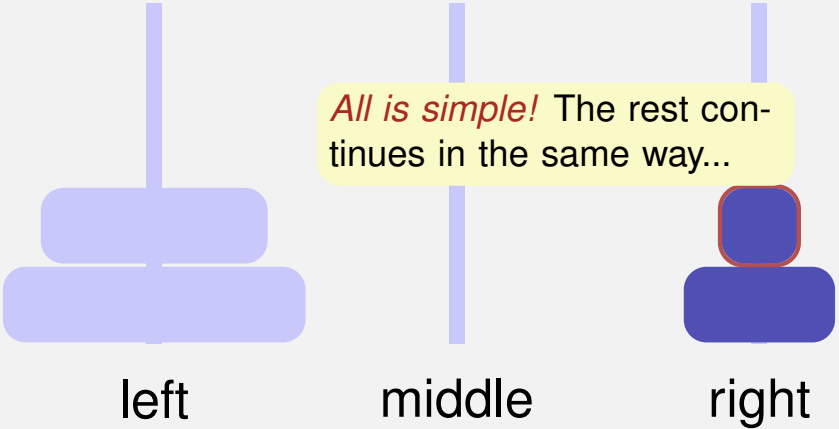
The Towers of Hanoi – Recursive Approach



The Towers of Hanoi – Recursive Approach



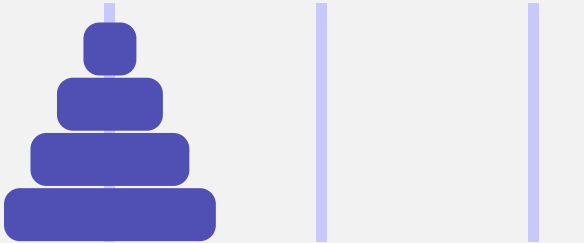
The Towers of Hanoi – Recursive Approach



Exercise Towers of Hanoi

- Open "Towers of Hanoi" on **code** expert
- Think about how you would approach the problem with pen and paper
- Implement a solution (optionally in groups)

The Towers of Hanoi – Code



Move 4 discs from `left` to `right` with auxiliary staple `middle`:

```
move(4,"left","middle","right")
```

The Towers of Hanoi – Code

`move(`n`,` src`,` aux`,` dst`)` $\Rightarrow$

1. Move the top $n-1$ discs from src to aux with auxiliary staple dst:
   `move(`$n-1$`,` src`,` dst`,` aux`);`
2. Move 1 disc from src to dst
   `move(1,` src`,` aux`,` dst`);`
3. Move the top $n-1$ discs from aux to dst with auxiliary staple src:
   `move(`$n-1$`,` aux`,` src`,` dst`);`

The Towers of Hanoi – Code

```
void move(int n, const string &src, const string &aux, const string &dst){
    if (n == 1) {
        // base case (‘move’ the disc)
        std::cout << src << " --> " << dst << std::endl;
    } else {
        // recursive case



    }
}
```

The Towers of Hanoi – Code

```
void move(int n, const string &src, const string &aux, const string &dst){
    if (n == 1) {
        // base case (‘move’ the disc)
        std::cout << src << " --> " << dst << std::endl;
    } else {
        // recursive case
        move(n-1, src, dst, aux);


    }
}
```

The Towers of Hanoi – Code

```
void move(int n, const string &src, const string &aux, const string &dst){
    if (n == 1) {
        // base case (‘move’ the disc)
        std::cout << src << " --> " << dst << std::endl;
    } else {
        // recursive case
        move(n-1, src, dst, aux);
        move(1, src, aux, dst);

    }
}
```

The Towers of Hanoi – Code

```
void move(int n, const string &src, const string &aux, const string &dst){
    if (n == 1) {
        // base case (‘move’ the disc)
        std::cout << src << " --> " << dst << std::endl;
    } else {
        // recursive case
        move(n-1, src, dst, aux);
        move(1, src, aux, dst);
        move(n-1, aux, src, dst);
    }
}
```

The Towers of Hanoi – Code

```
void move(int n, const string &src, const string &aux, const string &dst){
    if (n == 1) {
        // base case ('move' the disc)
        std::cout << src << " --> " << dst << std::endl;
    } else {
        // recursive case
        move(n-1, src, dst, aux);
        move(1, src, aux, dst);
        move(n-1, aux, src, dst);
    }
}
int main() {
    move(4, "left", "middle", "right");
    return 0;
}
```

The Towers of Hanoi – Code Alternative

```
void move(int n, const string &src, const string &aux, const string &dst){
    // base case
    if (n == 0) return;

    // recursive case
    move(n-1, src, dst, aux);
    std::cout << src << " --> " << dst << "\n";
    move(n-1, aux, src, dst);
}


int main() {
    move(4, "left", "middle", "right");
    return 0;
}
```

Questions?

4. Outro

General Questions?

See you next time

Have a nice week!